(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
1 February 2001 (01.02.2001)

PCT

(10) International Publication Number
WO 01/08098 A1

(51) International Patent Classification[7]: G06T 5/00

(21) International Application Number: PCT/US00/19900

(22) International Filing Date: 20 July 2000 (20.07.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/357,723 21 July 1999 (21.07.1999) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 09/357,723 (CON)
Filed on 21 July 1999 (21.07.1999)

(71) Applicant *(for all designated States except US)*: **OBVIOUS TECHNOLOGY, INC.** [US/US]; Suite 303, 580 Howard Street, San Francisco, CA 94015 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **PANCHANATHAN, Sethuraman** [CA/CA]; 480 S. Meadows Drive, Chandler, AZ 85224 (US). **JAMA, Ismail** [CA/CA]; Station B, P.O. Box 566, Ottawa, Ontario K1P 5P7 (CA).

(74) Agents: **PRATT, John, S.** et al.; Kilpatrick Stockton LLP, Suite 2800, 1100 Peachtree Street, Atlanta, GA 30309-4530 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ,

[Continued on next page]

(54) Title: OBJECT EXTRACTION IN IMAGES

(57) Abstract: A method and apparatus for extracting objects from an input image allows a user to select seed coordinates of the desired object. An edge detection component detects an edge of the object by comparing the intensity of a pixel to its surrounding pixels. A seeded region grower creates a segmented object image by converting the input image and the edge-detected image into gray scale data and further binarizing the edge-detected gray scale data. The seeded region grower creates a base region beginning with the seed coordinates and grows this region until it includes an outline of the object. A containment component then maps the segmented object image in four separate directions to tag pixels located in the desired object. An output component creates the output image containing the object by selecting the tagged pixels from the original input image.

WO 01/08098 A1

DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) **Designated States** *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*

(48) Date of publication of this corrected version:
12 April 2001

(15) Information about Correction:
see PCT Gazette No. 15/2001 of 12 April 2001, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
1 February 2001 (01.02.2001)

PCT

(10) International Publication Number
WO 01/08098 A1

(51) International Patent Classification[7]: G06T 5/00

(21) International Application Number: PCT/US00/19900

(22) International Filing Date: 20 July 2000 (20.07.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/357,723 21 July 1999 (21.07.1999) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US Not furnished (CON)
Filed on Not furnished

(71) Applicant *(for all designated States except US)*: OBVIOUS TECHNOLOGY, INC. [US/US]; Suite 303, 580 Howard Street, San Francisco, CA 94015 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: PANCHANATHAN, Sethuraman [CA/CA]; 480 S. Meadows Drive, Chandler, AZ 85224 (US). JAMA, Ismail [CA/CA]; Station B, P.O. Box 566, Ottawa, Ontario K1P 5P7 (CA).

(74) Agents: PRATT, John, S. et al.; Kilpatrick Stockton LLP, Suite 2800, 1100 Peachtree Street, Atlanta, GA 30309-4530 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR,

[Continued on next page]

(54) Title: OBJECT EXTRACTION IN IMAGES

(57) Abstract: A method and apparatus for extracting objects from an input image allows a user to select seed coordinates of the desired object. An edge detection component detects an edge of the object by comparing the intensity of a pixel to its surrounding pixels. A seeded region grower creates a segmented object image by converting the input image and the edge-detected image into gray scale data and further binarizing the edge-detected gray scale data. The seeded region grower creates a base region beginning with the seed coordinates and grows this region until it includes an outline of the object. A containment component then maps the segmented object image in four separate directions to tag pixels located in the desired object. An output component creates the output image containing the object by selecting the tagged pixels from the original input image.

a

b

WO 01/08098 A1

LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) **Designated States** *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*
— *Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

OBJECT EXTRACTION IN IMAGES

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention is directed to computer image processing, and more particularly, to object recognition and extraction in computer images.

2. Background

Image processing takes images from the real world--captured by cameras, infra-red sensors, ultrasound scanners, or other devices--and manipulates those images to achieve a desired result. Image processing is used in many applications, including the remote sensing of images beamed back from satellites, machine vision of faulty parts on a production line, and the enhancement and analysis of scanned images of the body. Recently, image processing has been incorporated into a range of more mainstream computing applications, such as desktop publishing and multimedia. Multimedia techniques such as video image compression, enhancement, and warping all require image processing techniques.

The objective of image processing is to visually enhance or statistically evaluate some aspect of an image not readily apparent in its original form. To that end, object recognition or object extraction becomes extremely important. Object recognition is a system's attempt to find an object within a particular image. The system rids the image of all elements that do not fit the system's definition of an object, compares the remaining image to the system's definition, and decides whether to recognize the remainder of the image as an object. Thus, automated object recognition consists of two tasks: (1) extracting relevant information from an image; and (2) making decisions about that information.

In multimedia applications, object extraction is extremely important in indexing and accessing images quickly. For instance, news agencies frequently store footage of news stories in video databases. In order to properly index the images, the agency must have knowledge of the subjects contained in those images. One method for indexing the images would be to manually search the footage and divide it according to subject matter. Unfortunately, such a method would be very tedious and time-consuming. A faster approach

would involve the use of image processing. A user would select a particular object and then retrieve all images containing that object.

Despite its advantages, however, object extraction is extremely difficult. Most image processing works with gray-scale images. Thus, a system begins with a two-dimensional array of pixels, each of which has an integer value representing gray scale, or brightness. From there, the image processor can only discard information, it cannot meaningfully add information to what it has received. The complexities of image processing arise in determining which information is and is not relevant and how to retain the data important to making a decision about a given object. Moreover, the overlapping of objects makes multiple object extraction difficult as well.

Several techniques have been developed to handle the complexities of object extraction. The two most common methods used in object recognition and extraction are: (1) region growing; and (2) edge detection. Region growing is an iterative bottom-up approach to object recognition. During region growing, an initial seed pixel is chosen to be the base region where the iteration will commence. Thereafter, for each of the eight pixels surrounding the seed pixel, the image processor decides whether to include the pixel with the seed pixel as part of the object. This process is repeated for every pixel within the image. Region growing is a limited process, however. Very often, when dealing with small regions, the region growing algorithm mistakenly attributes neighboring pixels to different objects. Thus, the algorithm fails to extract the proper object from the image.

An alternative approach to region growing is edge detection. Edge detection is a top-down method for extracting objects. During edge detection, an image processor extracts an object by discerning significant changes between the boundaries of various objects. The processor examines factors such as the intensity value of a pixel and compares that value to a neighboring pixel. Edge detection poses additional problems, however, such as the failure to distinguish between different portions of the object. This failure results in non-object items being extracted along with the desired object.

Moreover, both region growing and edge detection lack sufficient user input. With both methods, the user selects an image and then allows an algorithm to extract a desired object. During processing, the user cannot assist in the extraction of the object or offer any other meaningful input. In many instances, the user may be able to quickly assist in the

extraction process by specifying particular portions of an object within an image and/or eliminating certain undesired portions of an extracted object.

SUMMARY OF THE INVENTION

The present invention overcomes the problems of the prior art by incorporating region growing, edge detection, and a new containment process. The containment process tags particular pixels from a segmented object image. An output component then creates a new image using the tagged pixels from the original image. The containment processes limits the effects of holes within the extracted object. In addition, the present invention allows the user to assist in the extraction process.

In accordance with the purpose of the invention, as embodied and broadly described herein the invention is software for extracting an object from an input image. The software receives seed coordinates of the object within the input image from a user. The system then creates an edge-detected image from the input image using edge detection techniques. The input image is then converted into gray scale data. In addition, data from the edge-detected image is converted into edge-detected gray scale data. The software then binarizes the edge-detected gray scale data.. Next, the software creates a segmented object image using the seed coordinates, the gray scale data from the input image, and the binarized edge-detected gray scale data. The segmented object image is then mapped in four directions to determine whether a pixel belongs to an outline of the object or to a background element. The pixels belonging to the outline of the object are stored. Finally, an output image is created from the input image based on the location of the pixels belonging to the outline of the object.

In further accordance with the purpose of this invention, as embodied and broadly described herein the invention is a computer system having a memory comprising: a seed coordinate receiving component that receives seed coordinates of the object; an edge detection component that detects an edge of the input image; an image binarization component that binarizes the edge-detected image; a threshold computation component that computes a threshold for a region growing component; a region growing component that creates a segmented object image using the seed coordinates, data from the edge-detected image and data from the input image; a containment component that maps the segmented object image in at least one direction to determine whether a pixel belongs to an outline of

the object or to a background element and stores a location of the pixels belonging to the outline of the object; and an output component that creates an output image from the input image based upon the pixels determined to belong to the outline of the object.

Objects and advantages of the invention will be set forth in part in the description which follows and in part will be obvious from the description or may be learned by practice of the invention. The objects and advantages of the invention will be realized and attained by means of the elements and combinations particularly pointed out in the appended claims.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a block diagram of a computer system, in accordance with a preferred embodiment of the present invention.

FIGURE 2 is a block diagram of the components of the image processor of the present invention.

FIGURES 3A & 3B are a flow chart showing steps performed by the main component of the image processor of FIGURE 1.

FIGURE 4 is an illustration of a 3 X 3 group of pixels in an input image.

FIGURE 5 is a flow chart describing steps performed by the edge detector of FIGURE 2.

FIGURE 6 is a flow chart describing steps performed by an image input subroutine.

FIGURE 7 is a flow chart illustrating steps performed within a threshold computation routine.

FIGURE 8 is a flow chart illustrating steps performed by region grower

FIGURE 9 is a flow chart illustrating steps performed by the containment component.

FIGURE 10 is a flow chart illustrating steps performed within the image output routine.

FIGURES 11a and 11b are illustrations of an image containing an object to be extracted.

DETAILED DESCRIPTION

Reference will now be made in detail to the invention, examples of which are illustrated in the accompanying drawings. Wherever possible, the same reference numbers will be used throughout the drawings to refer to the same or like parts.

The present invention is an apparatus and method for extracting an object from an image. Although the present invention is illustrated as software, it may also be implemented as a hardware component in, for example, a graphics card in a computer system. FIGURE 1 is a block diagram of a computer system 100, in accordance with the present invention. Computer system 100 includes a CPU 102, a memory 104, and input/output lines 106; an input device 150, such as a keyboard or mouse; and a display device 160, such as a display terminal. Computer system 100 can also include numerous elements not shown for the sake of clarity, such as disk drives, network connections, additional memory, additional CPUs, etc.

Memory 104 includes a source image file 110 containing an object to be extracted, an image processor 111 for extracting the object, and a target image file 124 that will contain the extracted object. FIGURES 11a and 11b are examples of an image 1100 that may be contained in image source file 110. The image 100 includes a beaver object 1105 and a tree object 1110. For the purposes of this discussion, the user desires to extract the beaver object 1105 from the image 1100. FIGURE 11b shows the extracted beaver object. The image processor 111 is preferably a Unix command-line executable C programming language program, containing several components. It should be apparent that other programming languages may be used.

FIGURE 2 is a block diagram of image processor 111. Image processor 111 contains several components, including an image file converter component 112, an image input component 124, an edge detector component 114, a seeded region grower component 116, a threshold computation component 117, a containment component 118, and an image output component 126. Each component of Image processor 111 contains sub-components that will be described in greater detail below. Image processor 111 includes sub-components, as well, such as a main routine. Image processor 111 inputs source image file 110 containing the desired object, extracts the object from the image source file 110 and outputs the extracted object into target image file 124. All parts of image processor 111, preferably, are embodied

as instructions stored in memory 104 and performed by CPU 102 although these components may also be embodied in hardware. Image processor 111 may be embodied as instructions stored on removable storage medium 92 which is read by drive 90. Memory 104 also contains additional information, such as application programs, operating systems, data, etc., which are not shown in the figure for the sake of clarity. A preferred embodiment of the invention runs under the Solaris operating system, Version 2.3 and higher. Solaris is a registered trademark of Sun Microsystems, Inc. Other operating systems and/or environments may be used as well.

As stated above, image processor 111 is preferably a command-line-based executable application. As such the application may accept arguments from the command line. The image processor 111 accepts the following program arguments from a user at the command line: (1) "-x (X)(Y)", where X and Y are seed coordinates of the desired object within source image file 110; (2) "-c (R)(G)(B)", where the user may assign the RGB (red, green, blue) value of the background of the target image file (the image processor 111, preferably, defaults to a background color, such as white, where (R)=(G)=(B)=255); and (3) "-a (0/1)", where "0" is chosen if the user desires the extraction of a single object and "1" is chosen if the user desires the extraction of multiple objects.

All C programming language programs begin in a "*main*" subroutine. The *main* subroutine is then responsible for calling other subroutines in the program. FIGURE 3 is a flow chart showing steps performed within the *main* subroutine of image processor 111. The *main* subroutine of image processor 111 is responsible for accepting the command-line program arguments from the user as described above, performing various checks to ensure the validity of those program arguments, and calling other subroutines and components of image processor 111 to perform input of the source image file 110, extraction of the desired object, and output of the target file 124 containing the extracted image. In a first step 302, image processor 111 reads in the user-entered command-line program arguments (including the seed coordinates) and stores these arguments in memory 104. Next, in step 304, image processor 111 verifies that the command line arguments are valid arguments by comparing the arguments to a list of stored arguments. If the arguments are not valid, the image processor immediately halts execution in step 306. The image processor 111 may also provide an indication of invalidity to the user by, for example, displaying "Error: Invalid

Input Parameters" on display device 160. If the arguments are valid, processing continues at step 308. In step 308, the image processor 111 allocates a block of memory for a plurality of dynamic arrays that will be used by the various components of the image processor 111. Specifically, image processor 111 creates a two-dimensional *gray_array* array, a two-dimensional *color_array* array, a *header_array* array, a *boundary_array* array. The main subroutine then calls image file converter component 112 in step 312.

Image file converter component 112 converts images from one file format to another. Among the various image file formats that may be converted are joint photographic experts group ("JPEG") format, graphic image file ("GIF") format, and bitmap ("BMP") format, although other formats may be converted. Image file converter 112 converts input source image file 110 from its original format to a portable pixmap ("PPM") format. Image file converter 112 then stores the converted input image in a temporary intermediate PPM image file 115. Image conversion is performed using techniques known in the art, such as those used in the software sold under the trademark "Image Magic" by Eastman Kodak Company. Other image conversion tools may also be used to convert input image source file 110 to PPM format.

Once image converter 112 has converted the image source file 110 to the PPM format, image converter 112 returns an intermediate PPM image 115 to the *main* subroutine. In step 316, the *main* subroutine edge-detects the image by calling the edge detector component 114. Edge detector component 114 detects the edge of input source image 110. Both the conversion of the image source file to PPM format and the edge-detection of the source file 110 may be performed in a parallel manner to improve processing speed. Edge detection is performed using a Laplacian method. In the Laplacian method, an examined pixel is considered to be the center of a 3 X 3 window of pixels. FIGURE 4 is an illustration of a 3 X 3 window 400 of pixels that create an image. The examined pixel 401 lies in the center of the window 400. Since the local variance is larger near the boundary regions of the images, the calculated variance over the tested window 400 may be used to determine the resolution at a given pixel. Based on the determined local variance, a center test pixel 401 with a similar intensity value to those its neighboring pixels in the window 400 is determined to be homogeneous at that resolution. Since the noise in the uniform region is more

objectionable than in the boundary region, the noise elements in the uniform region must be suppressed or smoothed.

FIGURE 5 is a flow chart describing the edge detection process performed by the edge detector component 114. In step 502, edge detector component 114 enters a double nested loop to edge-detect every pixel contained in the image. The double nested loop allows the edge detector component 114 to traverse horizontal and vertical coordinates within an image. In step 506, within the loops, the local mean is estimated. Next, in step 508, the local variance is calculated over the window 300. The local variance is assumed to have an expected value that is smaller than the peak of its distribution calculated over an entire image, where the distribution is assumed to be unimodal. Each pixel is then assigned an integer value according to the comparison results. The integer value represents a threshold level used to estimate the presence of an edge. Each time a test is conducted to detect the presence of an edge, the average mean of that pixel in relation to its neighbors is computed and compared to this threshold The edge detector 114 repeats this process for every pixel in the image. When all of the pixels have been assigned an integer value, edge detector 114 creates an intermediate edge-detected image 119. The edge detector 114 returns the edge-detected image 119 to the *main* subroutine. The edge detection process may be performed by an edge detection component of the Image Magic software described above. The edge-detected image is converted to PPM format to create an edge-detected PPM image file. FIGURE 10 illustrates the results of the edge detection on the image 900.

Once control returns to *main*, the *main* subroutine then passes control to the image input component 121 in step 320. The image input component 121 performs several tasks. First, the component 121 stores the intermediate input PPM image 115 created by image file converter 112. Second, the header of the PPM image 115 is stored in an intermediate location (an array) and the image information stored in the header, including the image dimension, is extracted. Third, the raw RGB data is extracted from the remainder of the original PPM image. This raw data will be passed to several components for computation. Fourth, the image input component 121 converts the RGB color PPM image 115 into gray scale and stores the gray scale data into another array. Finally, the raw RGB data extracted from the original PPM image is passed to the edge detection component, where it is converted into gray scale data and binarized.

FIGURE 6 is a flow chart describing steps performed during the input image routine performed by image input component 121. In step 602, the routine opens input PPM image 115 and edge-detected PPM image 119 for reading. In step 604, the input component 125 reads the header of input PPM image 115 and edge-detected PPM image 119. In step 606, the input component 125 stores the read headers of both files into the *header_array* array.

Next, the image input component 125 enters two nested loops. The initialization statement of the first "outside" loop sets a loop control variable, *I*, equal to zero. The expression of the first loop (i.e., the conditional expression that determines whether or not the loop will repeat) requires the loop control variable, *I*, to be less than or equal to *X*, where *X* is the X-coordinate parameter of the original input image. Following each execution of the loop, the loop control variable is incremented by a value of 1. The initialization statement of the second "inside" loop sets a second loop control variable equal to zero. The expression of the second loop requires the second loop control variable, J, to be less than or equal to Y, where Y is the Y-coordinate parameter of the original input image. Within the loops, in step 610, the input routine reads red pixel values, green pixel values, and blue pixel values at the particular X and Y coordinate in the input PPM image 115. In step 612, the routine stores the RGB color pixel value in the array, *color_array (I,J)*, where I & J are the loop control variables for the first and second loops, respectively. Thus, I and J correspond to a pixel at a particular coordinate. Next, the input routine changes the RGB color pixel to a gray scale pixel using an image processing function. For example, the following function would provide a gray level value of an RGB pixel of 24 bits having a red component, a green component, and a blue component:

Gray_Level [R, G, B] = .299 * Color_image[R] + .587 * Color_image[G] + .114 * Color_image[B].

If Gray_Level[R, G, B] < 0, then Gray_Level[R, G, B] =0

Else Gray_Level[R, G, B] = the floor function of Gray_Level[R, G, B] +.5

In step 616, the component 125 stores the gray pixel value in the location *gray_array (I,J)*. In step 620, the component 125 reads the RGB values of the color edge-detected image. Next, the input component 125, in step 622 converts the color edge-detected image into gray scale data. Next, in step 624, the component 121 binarizes the edge-detected gray scale

pixel. Specifically, if the gray scale value of the edge-detected pixel is greater than 100, the pixel is given a binary value of 1. If the gray scale value of the edge-detected pixel is less than 100, the pixel is given a value of 0. The routine then stores the binarized pixel values from the edge-detected image in the location *boundary_array (I,J)*. The binarization makes it easier to use the edge detection image in boundary recognition. The second loop is then closed followed by the first loop. In step 626, the input component 121 closes all of the opened files. The loops are shown in their entirety below:

```
For (I=0; I≤X; I++) {
        For (J=0; J≤Y;J++) {
                Read Red Pixel value of input image file;
                Read Green pixel value of input image file;
                Read blue pixel value of input image file;
                Store RGB color pixel value in the color_array (I,J);
                Convert to gray scale pixel (see above formula);
                Store gray pixel value in the location gray_array(I,J);
                Read Red Pixel value of edge-detected image file
                Read Green Pixel value of edge-detected image file
                Read Blue Pixel value of edge-detected image file
                Change to gray scale pixel;
                Binarize edge-detected gray scale pixel;
                Store gray binarized pixel value in the location boundary_array(I,J);
        }
}
Close all files
```

Next, the *main* routine, in step 324, calls threshold computation component 117. Threshold computation component 117 computes a threshold value that will be used by the seeded region growing component 116 during the seeded region growing of the image. Region growing is an iterative approach, wherein a decision is made regarding the inclusion or exclusion of each pixel in the image. An initial seeded pixel centered in a 3 X 3 region of pixels is chosen to be the base region where the iteration will commence. Thereafter, the

software decides whether each of the 8 connected neighboring test pixels in the current iteration will be either combined with the base region or excluded from the base region. Each test pixel is examined from two perspectives: (1) a boundary examination to determine whether a boundary exists in each pixel; and (2) a homogeneity examination to determine if the pixel is similar to the base region. If both of these examinations have succeeded, the tested pixel is merged with the base region. "Homogeneity" represents the level of color or light intensity in the region. If the absolute difference of the mean of the test pixel value is within a pre-specified "threshold" of the mean of the base region, the test pixel is considered to be part of the desired object and, therefore, part of the base region.

FIGURE 7 is a flow chart illustrating how the threshold computation component 117 determines the threshold. In particular, the component 117 computes the variance of the input image and multiplies the variance by a predetermined constant variable, M. M is a constant value, where $0 < M \leq 1$. M is preferably derived from simulations on a variety of training images. The value of M affects the degree of object segmentation and, therefore, the quality of the segmented object. At the outset of the routine, in step 702, the variables *Mean*, *Variance*, and *Temporary*, are all set equal to 0. The loop proceeds in steps 706-708 as follows:

```
For (I=0; I≤X dimension; I++) {
        For (J=0; J≤Y dimension; J++) {
                Mean = Mean + gray_array(I,J)
                Temporary = Temporary + gray_array(I,J) * gray_array(I,J);
        }
}
```

The value stored in the array, *gray_array* is derived from the image input routine described above. Specifically, the array stores the gray scale value of the pixel. From the foregoing loop, the variable *Mean* is equal to the sum of the array, *gray_array*. A temporary variable is used in the calculation of the variance. The actual mean is then calculated in step 710 by dividing by the number of array elements:

Mean = *Mean*/(X dimension * Y dimension)

Similarly, in step 712, the temporary variable is divided by the total number of array elements:

Temporary = *Temporary*/(X dimension * Y dimension)

Next, in step 714, the variance is calculated as follows:

$$Variance = \sqrt{(Temporary - Mean * Mean)}$$

Finally, in step 716, the threshold is determined:

Threshold = M * *Variance*

Once the threshold has been computed, the *main* routine passes control to region grower component 116. Region grower performs a recursive subroutine responsible for the task of extracting the desired object. The seed region is initially supplied to the component when the extraction process begins. At the beginning of the segmentation process, the user specifies the seed region by, for example, clicking on a position within the object to be segmented. The program translates the mouse click to aX-seed and Y-seed coordinates. This translation may be performed using standard scripting languages. The region grower 116 performs a recursive check on neighboring regions or pixels for homogeneity. If the neighboring regions are homogeneous, they are added to the grown region. The boundary of the neighboring region is also checked for object boundary. Since the process is recursive, the region grower 116 includes a tagging mechanism to ensure that processed pixels are not re-processed. If the absolute difference of the mean of a tested pixel is within the pre-determined threshold of the mean of the value of the base region pixels, the tested pixel is considered to be part of the object and is added to the base region.

FIGURE 8 is a flow chart illustrating steps performed by region grower 116. In step 802, the region grower 116 initializes the elements of an array, *Tag (I,J)*, to zero. The array, *Tag (I,J)*, is used to ensure that processed pixels are not reprocessed within the recursive subroutine of the region grower 116. The array elements are initialized within two loops. The first loop that executes while the loop control variable, **I**, is less than or equal to the X dimension of the image. The second loop executes while the second loop control variable, **J**, is less than or equal to the Y dimension of the image. The loop proceeds, as follows:

```
For (I = 0; I≤X dimension; I++) {
        For (J = 0; J≤Y dimension; J++) {
                Tag (I,J) = 0;
```

```
        }
    }
```

Next, in step 804, the region grower 116 sets a variable, Total_Mean equal to the value of the array element, *gray_array (Xseed, Yseed)*, where *Xseed* and *Yseed* are the seed coordinates of the desired image, provided by the user at start-up. The array, *gray_array* was created by the image input subroutine, and contains the gray pixel values of the input image. In addition, region grower 116 sets a variable Count equal to 1.

The region grower 116 then enters a recursive subroutine, Segment, that performs the extraction of the image. The initial parameters to the segment routine are Xseed, Yseed, Total_Mean, and Count. Total_Mean divided by Count represents the average mean of the grown region. This will be compared to the value of the pixels under examination. Thus, extraction begins at the seed coordinates of the object. Thereafter, the routine calls itself with different coordinates. The routine executes only if the elements of *boundary_array (Xseed, Yseed)*, and *Tag (Xseed, Yseed)* are equal to zero. A value of one in the binarized edge-detected image would indicate that a boundary exists. Thus, the pixel is considered if its value is equal to zero. The array, *boundary_array*, was created by the image input subroutine and contains the binarized gray scale pixel of the edge-detected image. Next, region grower 116 sets the variable Mean equal to *gray_array (Xseed, Yseed)* and the variable Area_Mean is then set to Total_Mean/Count. A variable, Difference, is set equal to the absolute value of Area_Mean - Mean. Next, the region grower 116 determines whether the calculated Difference is less than the value of the Threshold as computed by the threshold computation component 117. If Difference is less than Threshold, the region grower 116 extracts the object by setting the pixel values of an array, segmented_object_array (Xseed, Yseed), equal to 1. The value of Total Mean is incremented by the value of Mean. The region grower 116 increments the Count variable and the tag of the subject pixel is set equal to 1. By setting the Tag elements equal to 1, the pixels will not be reprocessed. The region grower 116 then recursively calls itself to process each pixel within the desired object. The subroutine, Segment, is shown in its entirety below:

```
Segment (Xseed, Yseed, Total_Mean, Count) {
```

```
if (boundary_array (Xseed, Yseed) ==0) and (Tag (Xseed, Yseed) == 0)
{
        Mean = gray_array (Xseed, Yseed);
        Area_Mean = Total_Mean/Count;
        Difference = abs(Area_Mean-Mean);
        If (Difference < Threshold)
        {
                segmented_object_array (Xseed, Yseed) = 1;
                Total_Mean = Total Mean + Mean;
                Count++;
                Tag (Xseed, Yseed) = 1;

                if (Xseed > 0)
                { Segment (Xseed - 1, Yseed, Total_Mean, Count);}

                if (Xseed < X dimension)
                { Segment (Xseed + 1, Yseed, Total_Mean, Count);}

                if (Yseed > 0)
                { Segment (Xseed, Yseed - 1, Total_Mean, Count);}

                if (Yseed < 0)
                { Segment (Xseed, Yseed + 1, Total_Mean, Count);}
        }
    }
}
```

Once the pixels of the desired object have been extracted and stored in an array, *main* calls containment component 118 in step 328. Containment component 118 maps a segmented binary object image from four different directions. The segmented binary object image is an image resulting from assigning a binary value to each pixel in the image, where a

value of "1" indicates that the pixel is part of the desired object and a value of "0" indicates that the pixel is not part of the desired object. The object of the containment component is to tag each pixel of the extracted object as being either an outline of the object of the background image. The containment component outputs an array containing tagged pixels which will be used to construct the final extracted image. FIGURE 9 is a flow chart illustrating steps performed by the containment component 118.

In step 902, containment component 118 begins mapping in a first direction from X = 0 to X = X dimension. In particular, mapping is performed as follows: the component 118 sets X = to 0. Next, while X is less than or equal to the X dimension of the image, the pixel is tagged as part of the background if the pixel value is equal to zero. Otherwise, the boundary reached is set to true. The value of X is then incremented. In step 910, the containment component begins mapping in the reverse direction from X=X dimension to X = 0. X is initially set to X dimension. The mapping then proceeds until X = 0. Mapping in the Y direction is similar. Once the mapping is complete, the containment component has obtained the boundary coordinates of the object. Those coordinates are then used to replace the extracted object with the object from the original input image.

Following the mapping, *main* passes control to the image output component 126. The image output component 126 stores the output object image array into a temporary intermediate output PPM image. The image will be later converted into the appropriate format. In addition, the image output routine distinguishes between the extracted object pixels and background pixels. Background pixels are those taken either from that of a default value (white), a uniform background color supplied by the user, or the last extracted image pixels.

FIGURE 10 is a flow chart illustrating steps performed by the image output component 126. In step 1002. image output component 126 opens an output PPM file for writing. In step 1004, the component 126 opens a multiple extraction file for writing. In step 1006, the component 126 stores the headers from the PPM file from the header array. The component 126 then enters a loop to store RGB *color_array* in the temporary intermediate output PPM file opened in step 802. The loop is shown below:

```
For (I = 0; I≤X dimension; I++) {
        For (J = 0; J ≤ Y dimension; J++) {
```

```
If (segmented_object_array (I,J) is not tagged) {
        Store RGB color_Array (I,J) in the temporary intermediate
output Ppm;
        Store RGB color_array (I,J) in the temporary multiple extract
file}
else {
        Store RGB bg_array(I,J) in the temporary intermediate output
PPM;
        Store RGB bg_array (I,J) in the temporary multiple extraction
file}
    }
}
```

The image output component 126 then closes all of the files.

Having thus described a preferred embodiment of an image processor, it should be apparent to those skilled in the art that certain advantages have been achieved. It should also be appreciated that various modifications, adaptations, and alternative embodiments thereof may be made within the scope and spirit of the present invention. The invention is further defined by the following claims:

CLAIMS

What is Claimed is:

1. A method, performed by a computer system, for extracting an object from an input image comprising the steps of:

receiving seed coordinates of the object within the input image from a user;

creating an edge-detected image from the input image;

converting data from the input image into gray scale data;

converting data from the edge-detected image into edge-detected gray scale data;

binarizing the edge-detected gray scale data;

creating a segmented object image using the seed coordinates, the gray scale data from the input image, and the binarized edge-detected gray scale data;

mapping the segmented object image in at least one direction to determine whether a pixel belongs to an outline of the object or to a background element;

storing a location of the pixels belonging to the outline of the object; and

creating an output image from the input image based on the location of the pixels belonging to the outline of the object.

2. The method for extracting an object from an input image, as recited in Claim 1, wherein the step of creating an edge-detected image further comprises the steps of:

selecting a pixel within the input image;

comparing a resolution of the selected pixel to a resolution of a plurality of adjacent pixels; and

assigning a value to the selected pixel based on the comparison.

3. The method for extracting an object from an input image, as recited in Claim 1, wherein the step of creating a segmented image further comprises the steps of:

creating a base region beginning with a pixel at the seed coordinates;

analyzing additional pixels for homogeneity with neighboring pixels; and

growing the base region by adding homogeneous pixels to the base region.

4. The method for extracting an object from an input image, as recited in Claim 3, wherein the step of creating a segmented image further comprises the steps of:

computing a variance of the input image; and

multiplying the variance by a constant variable to determine a threshold value;

wherein a pixel is added to the base region if an absolute difference of a mean of the value of the pixel is less than the threshold value.

5. The method for extracting an object from an input image, as recited in Claim 1, wherein the step of mapping the segmented object image further comprises the steps of:

mapping the segmented object image in a first direction from a pixel having an X coordinate equal to zero to a pixel having an X coordinate equal to the greatest X coordinate in the segmented object image; and

mapping the segmented object image in a second direction from a pixel having an X coordinate equal to the greatest X coordinate in the segmented object image to a pixel having an X coordinate equal to zero.

6. The method for extracting an object from a computer-generated input image, as recited in Claim 5, wherein the step of mapping the segmented object image further comprises the steps of

mapping the segmented object image in a third direction from a pixel having a Y coordinate equal to zero to a pixel having a Y coordinate equal to the greatest X coordinate in the segmented object image; and

mapping the segmented object image in a fourth direction from a pixel having a Y coordinate equal to the greatest Y coordinate in the segmented object image to a pixel having a Y coordinate equal to zero.

7. The method for extracting an object from an input image, as recited in Claim 1, further comprising the step of converting the input image to a portable pixmat file format.

8. The method for extracting an object from an input image, as recited in Claim 7, further comprising the step of converting the output image to its original file format.

9. A method, performed by a computer system, for extracting an object from an input image comprising the steps of:

receiving seed coordinates of the object within the input image from a user;

creating an edge-detected image from the input image;

creating a segmented object image using the seed coordinates, data from the edge-detected image and data from the input image;

mapping the segmented object image in at least one direction to determine whether a pixel belongs to an outline of the object or to a background element;

storing a location of the pixels belonging to the outline of the object; and

creating an output image from the input image based on the location of the pixels belonging to the outline of the object.

10. The method for extracting an object from an input image, as recited in Claim 9, wherein the step of creating a segmented object image further comprises the steps of:

converting data from the input image into gray scale data;

converting data from the edge-detected image into edge-detected gray scale data; and

binarizing the edge-detected gray scale data.

11. The method for extracting an object from an input image, as recited in Claim 10, wherein the step of creating a segmented object image further comprises the steps of:

creating a base region beginning with a pixel at the seed coordinates;

analyzing additional pixels for homogeneity with neighboring pixels; and

growing the base region by adding homogeneous pixels to the base region.

12. The method for extracting an object from an input image, as recited in Claim 9, wherein the step of mapping the segmented object image further comprises the steps of:

mapping the segmented object image in a first direction from a pixel having an X coordinate equal to zero to a pixel having an X coordinate equal to the greatest X coordinate in the segmented object image; and

mapping the segmented object image in a second direction from a pixel having an X coordinate equal to the greatest X coordinate in the segmented object image to a pixel having an X coordinate equal to zero.

13. The method for extracting an object from an input image, as recited in Claim 12, wherein the step of mapping the segmented object image further comprises the steps of:

mapping the segmented object image in a third direction from a pixel having a Y coordinate equal to zero to a pixel having a Y coordinate equal to the greatest X coordinate in the segmented object image; and

mapping the segmented object image in a fourth direction from a pixel having a Y coordinate equal to the greatest Y coordinate in the segmented object image to a pixel having a Y coordinate equal to zero.

14. The method for extracting an object from an input image, as recited in Claim 13, wherein the step of storing a location of the pixels belonging to the outline of the object further comprises the steps of:

storing the location of the pixels along the first direction belonging to the outline of the object;

storing the location of the pixels along the second direction belonging to the outline of the object;

storing the location of the pixels along the third direction belonging to the outline of the object;

storing the pixels along the fourth direction belonging to the outline of the object.

15. The method for extracting an object from an input image, as recited in Claim 9, wherein the step of creating an output image from the input image based on the location of the pixels belonging to the outline of the object further comprises the steps of:

removing the pixels defined by the boundary of the stored locations from the input image;

placing the removed pixels in the output image at the stored locations.

16. The method for extracting an object from an input image, as recited in Claim 15, further comprising the steps of placing a plurality of default background pixels in the output image at locations other than the location of the object.

17. A computer system for extracting an object from an input image, the computer system having a memory, the computer system comprising:

a seed coordinate receiving component that receives seed coordinates of the object;

an edge detection component that detects an edge of the input image;

a region growing component that creates a segmented object image using the seed coordinates, data from the edge-detected image and data from the input image;

a containment component that maps the segmented object image in at least one direction to determine whether a pixel belongs to an outline of the object or to a background element and stores a location of the pixels belonging to the outline of the object; and

an output component that creates an output image from the input image based upon the pixels determined to belong to the outline of the object.

18. The computer system for extracting an object from an input image, as recited in Claim 17, further comprising an image converter that converts the input image into a different format.

19. The computer system for extracting an object from an input image, as recited in Claim 17, further comprising a threshold computation component that computes a threshold value used by the region growing component.

20. The computer system for extracting an object from an input image, as recited in Claim 19, wherein the region growing component further comprises:

means for creating a base region based upon the seed coordinates;

means for recursively determining a homogeneity of a plurality of pixels with a neighboring pixel;

means for adding homogeneous pixels to the base region to create a segmented object.

21. A computer program product, comprising:

a computer usable medium having a computer readable program code mechanism embodied therein configured to receive seed coordinates of an object within an input image from a user;

a computer readable program code mechanism configured to create an edge-detected image from the input image;

a computer readable program code mechanism configured to create a segmented object image using the seed coordinates, data from the edge-detected image and data from the input image;

a computer readable program code mechanism configured to map the segmented object image in at least one direction to determine whether a pixel belongs to an outline of the object or to a background element;

a computer readable program code mechanism configured to store a location of the pixels belonging to the outline of the object; and

a computer readable program code mechanism configured to create an output image from the input image based on the location of the pixels belonging to the outline of the object.

22. The computer program product of Claim 21, wherein the computer readable program code mechanism in the program product further comprises:

a computer readable program code mechanism configured to convert data from the input image into gray scale data;

a computer readable program code mechanism configured to convert data from the edge-detected image into edge-detected gray scale data; and

a computer readable program code mechanism configured to binarize the edge-detected gray scale data.

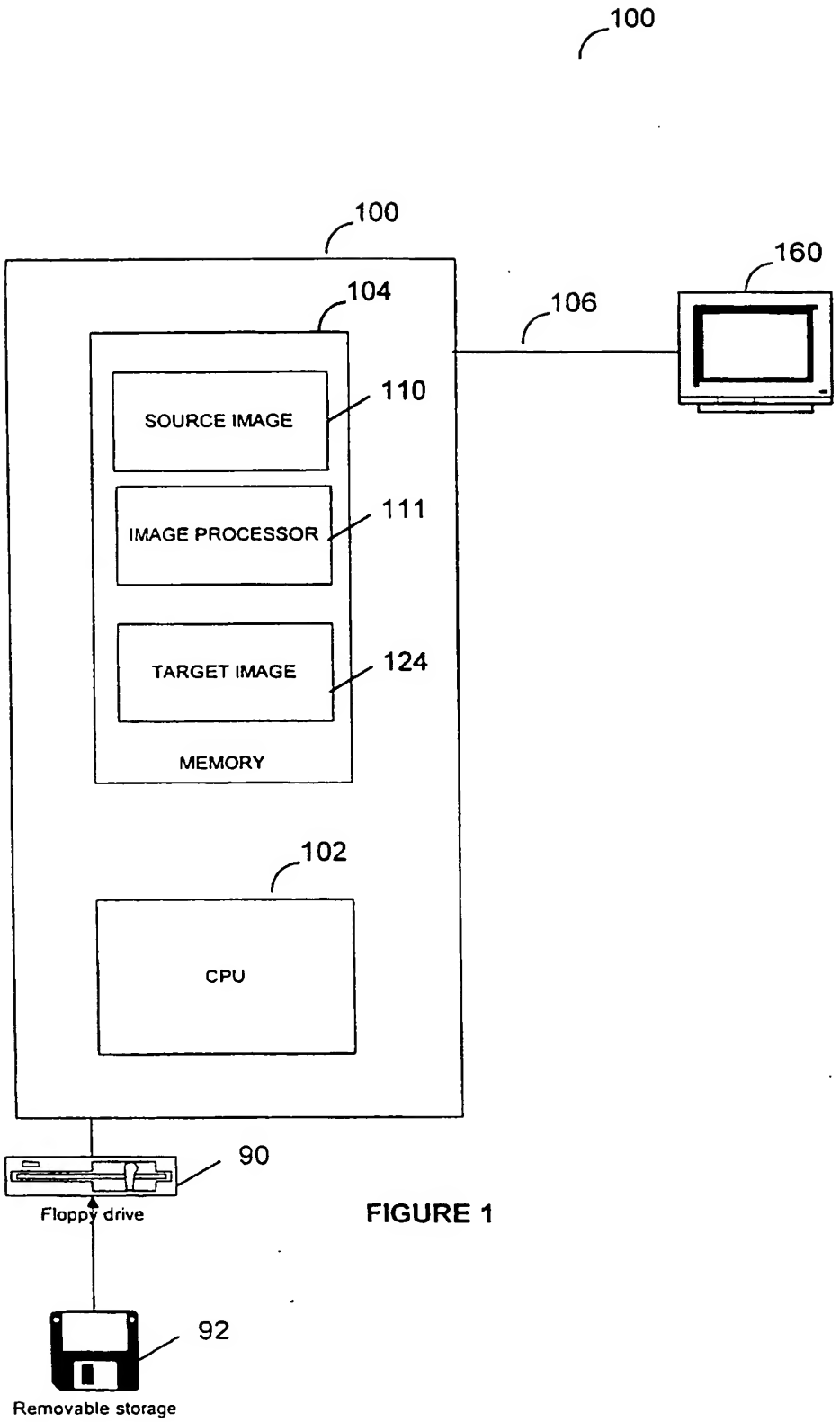
100
100
160
104
106
SOURCE IMAGE
110
IMAGE PROCESSOR
111
TARGET IMAGE
124
MEMORY
102
CPU
90
Floppy drive
92
Removable storage

FIGURE 1

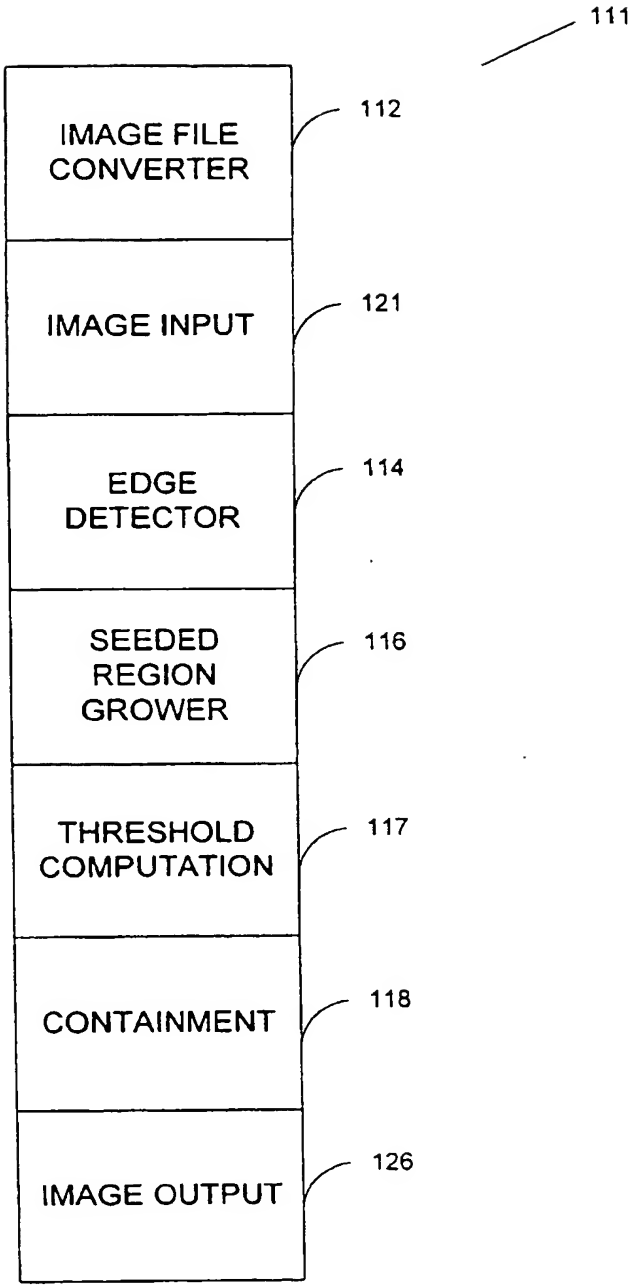
111
IMAGE FILE CONVERTER
112
IMAGE INPUT
121
EDGE DETECTOR
114
SEEDED REGION GROWER
116
THRESHOLD COMPUTATION
117
CONTAINMENT
118
IMAGE OUTPUT
126

FIGURE 2

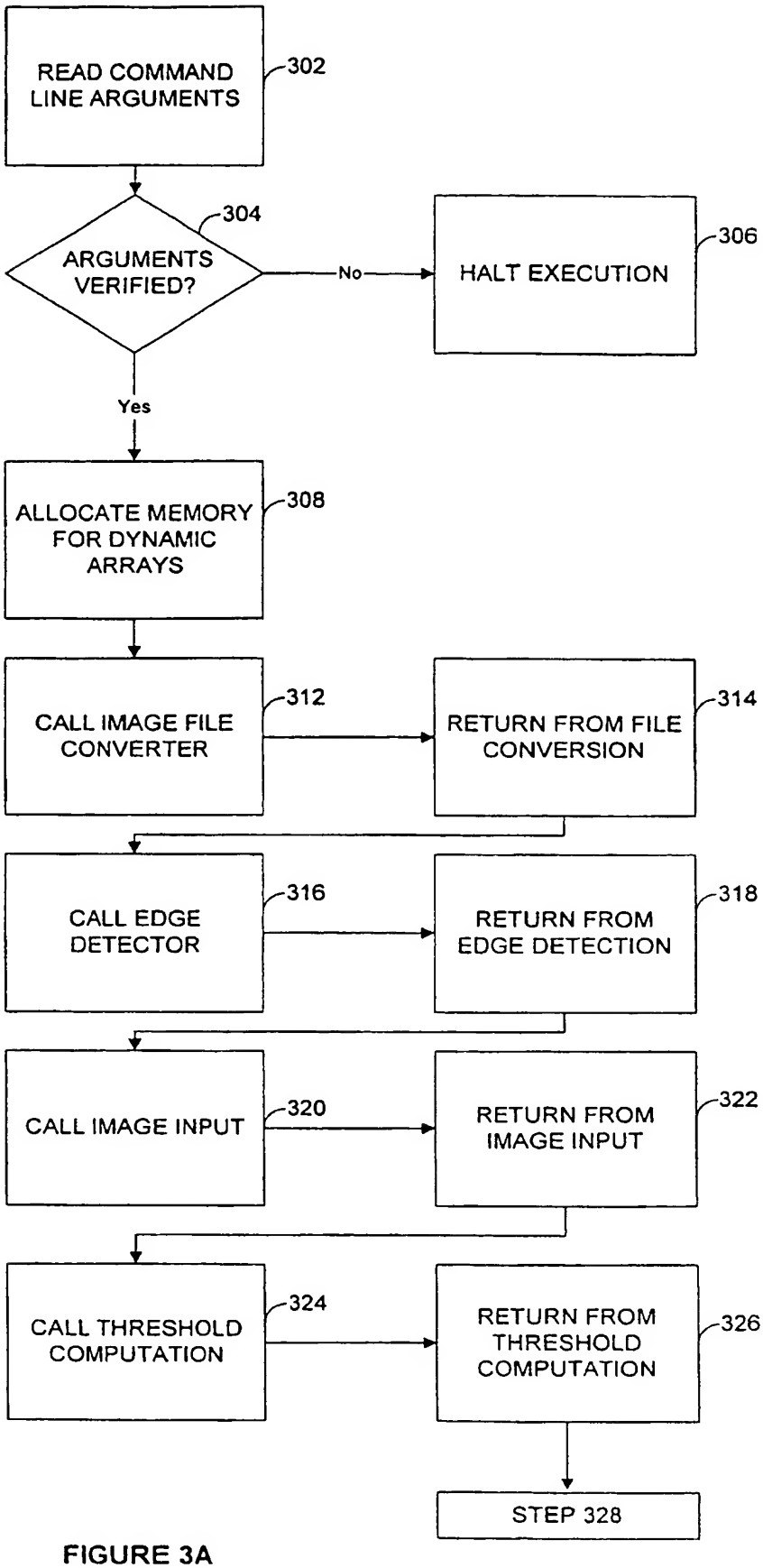
READ COMMAND LINE ARGUMENTS
302
ARGUMENTS VERIFIED?
304
No
HALT EXECUTION
306
Yes
ALLOCATE MEMORY FOR DYNAMIC ARRAYS
308
CALL IMAGE FILE CONVERTER
312
RETURN FROM FILE CONVERSION
314
CALL EDGE DETECTOR
316
RETURN FROM EDGE DETECTION
318
CALL IMAGE INPUT
320
RETURN FROM IMAGE INPUT
322
CALL THRESHOLD COMPUTATION
324
RETURN FROM THRESHOLD COMPUTATION
326
STEP 328

FIGURE 3A

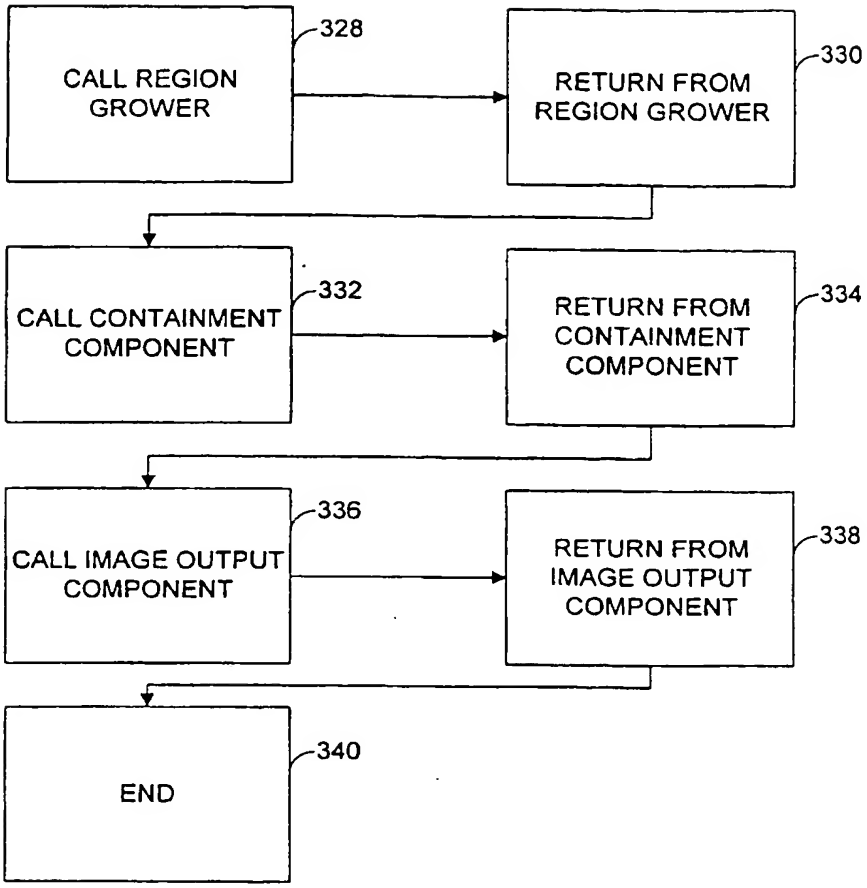
CALL REGION GROWER
328
RETURN FROM REGION GROWER
330
CALL CONTAINMENT COMPONENT
332
RETURN FROM CONTAINMENT COMPONENT
334
CALL IMAGE OUTPUT COMPONENT
336
RETURN FROM IMAGE OUTPUT COMPONENT
338
END
340

FIGURE 3B

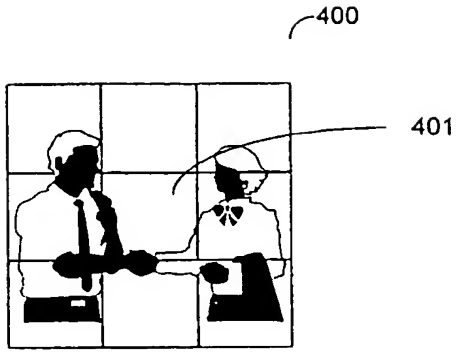

FIGURE 4

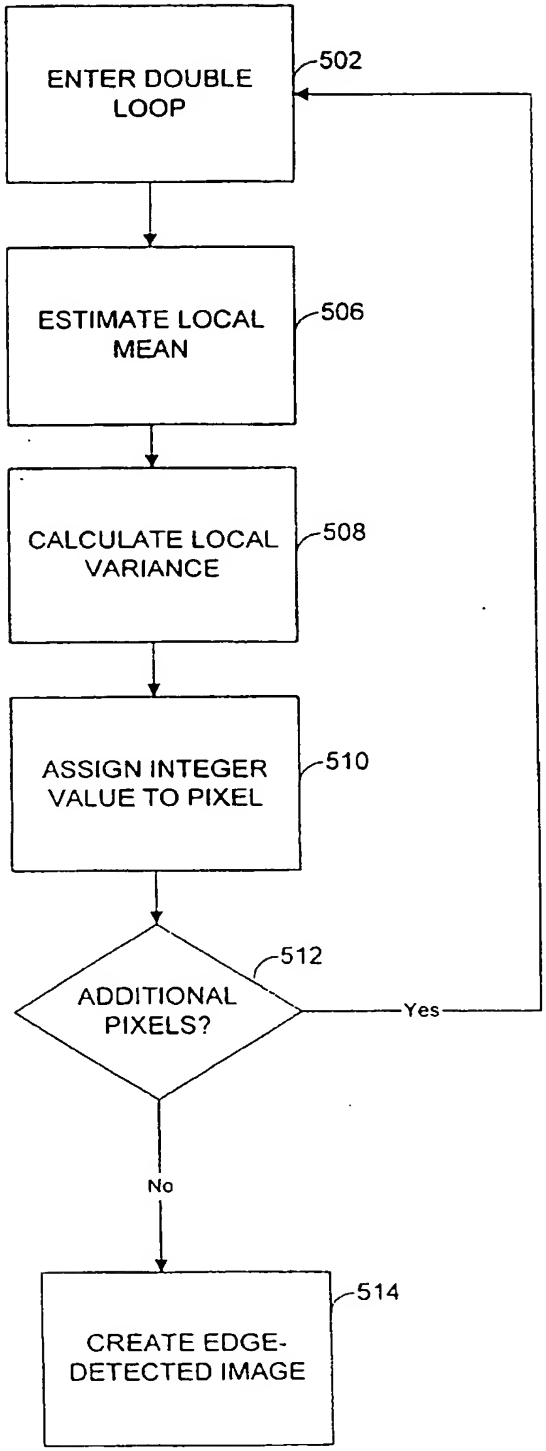
ENTER DOUBLE LOOP
502
ESTIMATE LOCAL MEAN
506
CALCULATE LOCAL VARIANCE
508
ASSIGN INTEGER VALUE TO PIXEL
510
ADDITIONAL PIXELS?
512
Yes
No
CREATE EDGE-DETECTED IMAGE
514

FIGURE 5

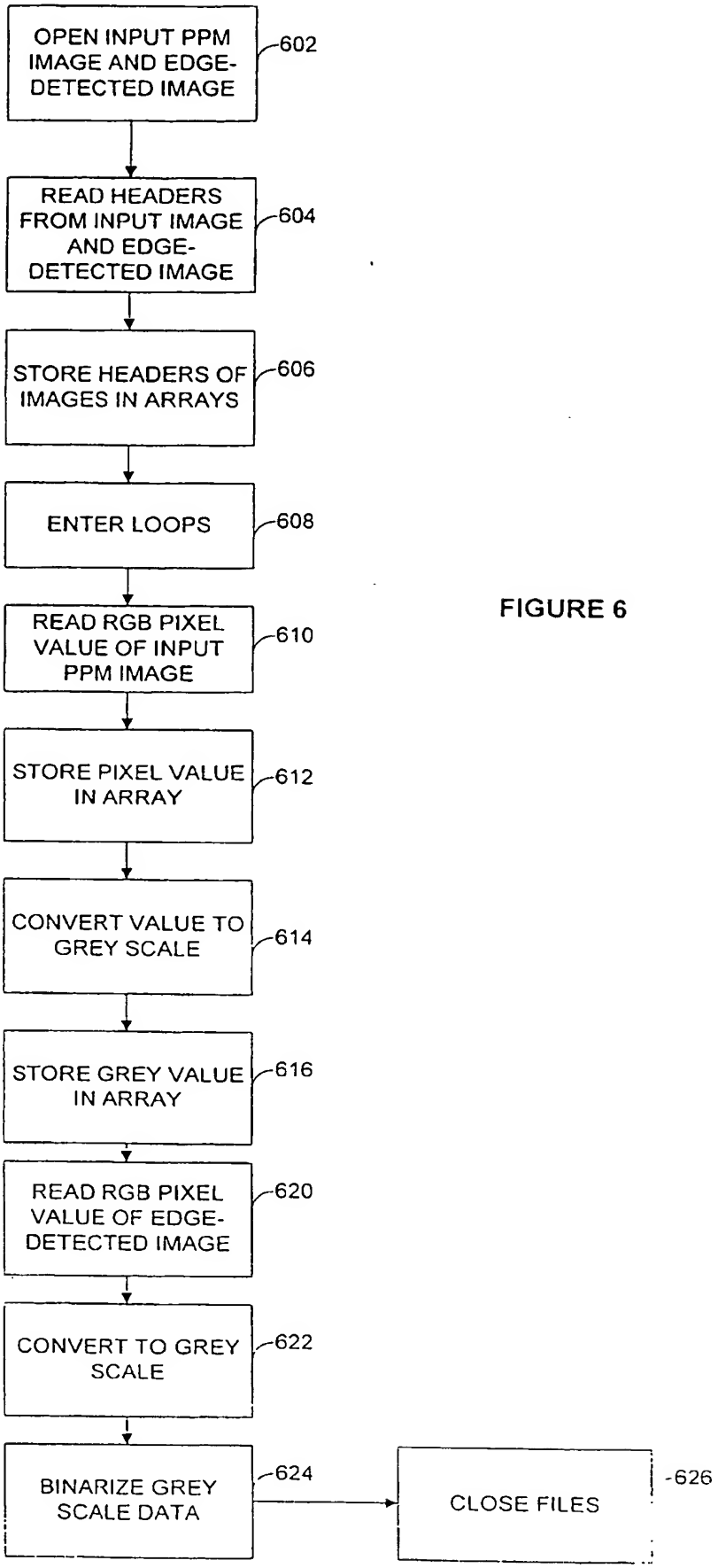
OPEN INPUT PPM IMAGE AND EDGE-DETECTED IMAGE
602
READ HEADERS FROM INPUT IMAGE AND EDGE-DETECTED IMAGE
604
STORE HEADERS OF IMAGES IN ARRAYS
606
ENTER LOOPS
608
READ RGB PIXEL VALUE OF INPUT PPM IMAGE
610
STORE PIXEL VALUE IN ARRAY
612
CONVERT VALUE TO GREY SCALE
614
STORE GREY VALUE IN ARRAY
616
READ RGB PIXEL VALUE OF EDGE-DETECTED IMAGE
620
CONVERT TO GREY SCALE
622
BINARIZE GREY SCALE DATA
624
CLOSE FILES
626

FIGURE 6

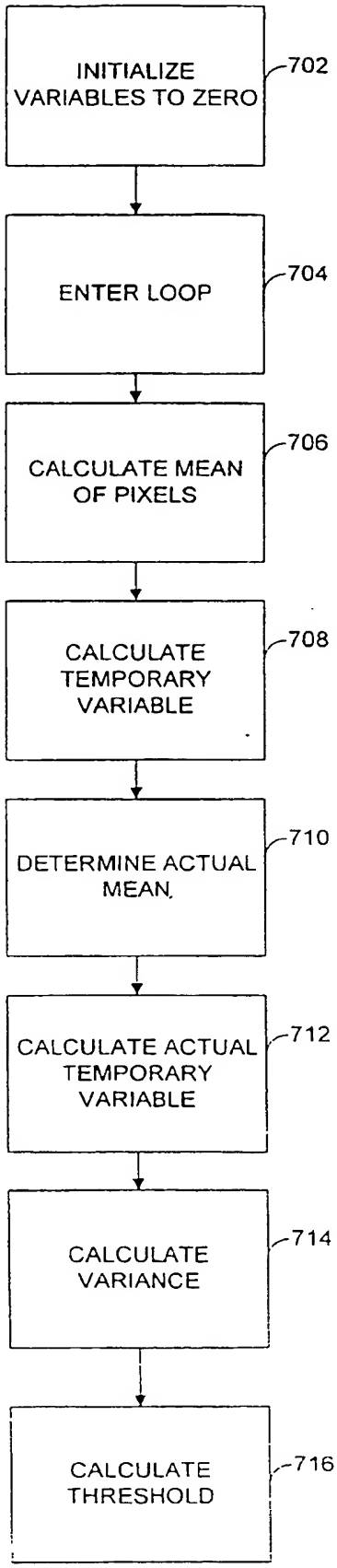
INITIALIZE VARIABLES TO ZERO
702
ENTER LOOP
704
CALCULATE MEAN OF PIXELS
706
CALCULATE TEMPORARY VARIABLE
708
DETERMINE ACTUAL MEAN
710
CALCULATE ACTUAL TEMPORARY VARIABLE
712
CALCULATE VARIANCE
714
CALCULATE THRESHOLD
716

FIGURE 7

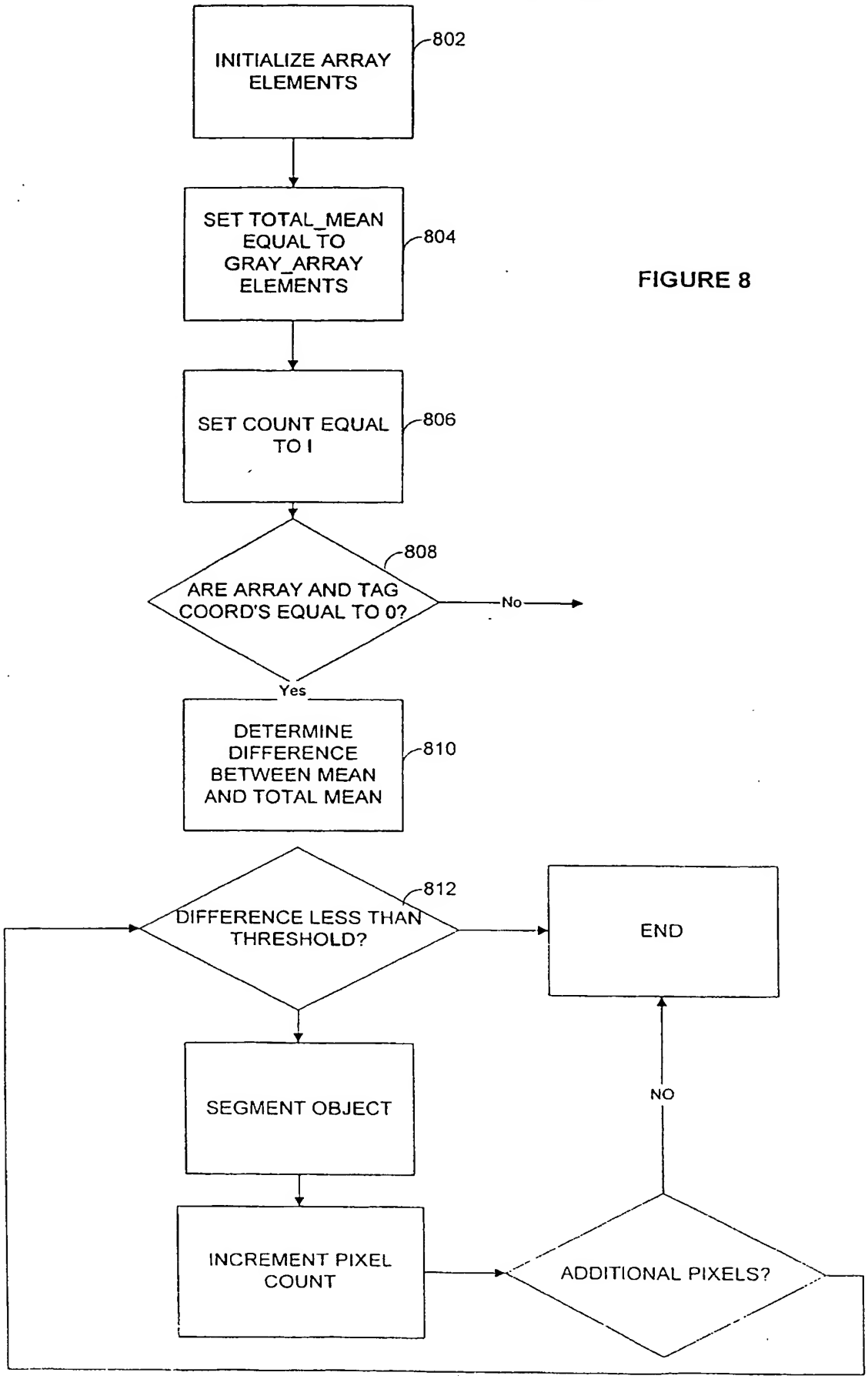
INITIALIZE ARRAY ELEMENTS
802
SET TOTAL_MEAN EQUAL TO GRAY_ARRAY ELEMENTS
804
SET COUNT EQUAL TO I
806
ARE ARRAY AND TAG COORD'S EQUAL TO 0?
808
No
Yes
DETERMINE DIFFERENCE BETWEEN MEAN AND TOTAL MEAN
810
DIFFERENCE LESS THAN THRESHOLD?
812
END
SEGMENT OBJECT
NO
INCREMENT PIXEL COUNT
ADDITIONAL PIXELS?

FIGURE 8

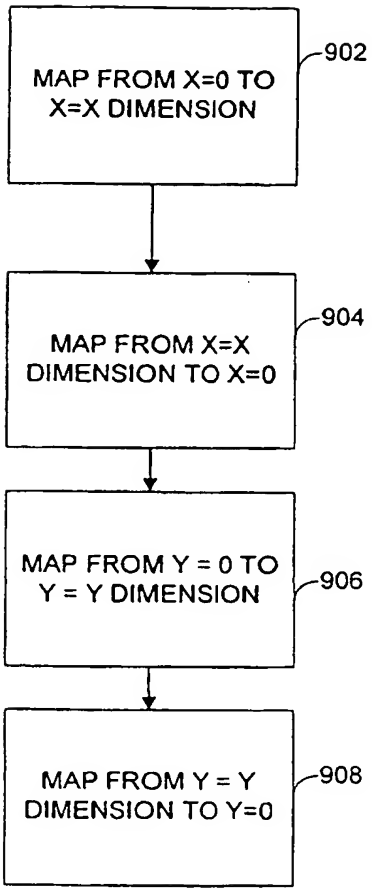
MAP FROM X=0 TO X=X DIMENSION
902
MAP FROM X=X DIMENSION TO X=0
904
MAP FROM Y = 0 TO Y = Y DIMENSION
906
MAP FROM Y = Y DIMENSION TO Y=0
908

FIGURE 9

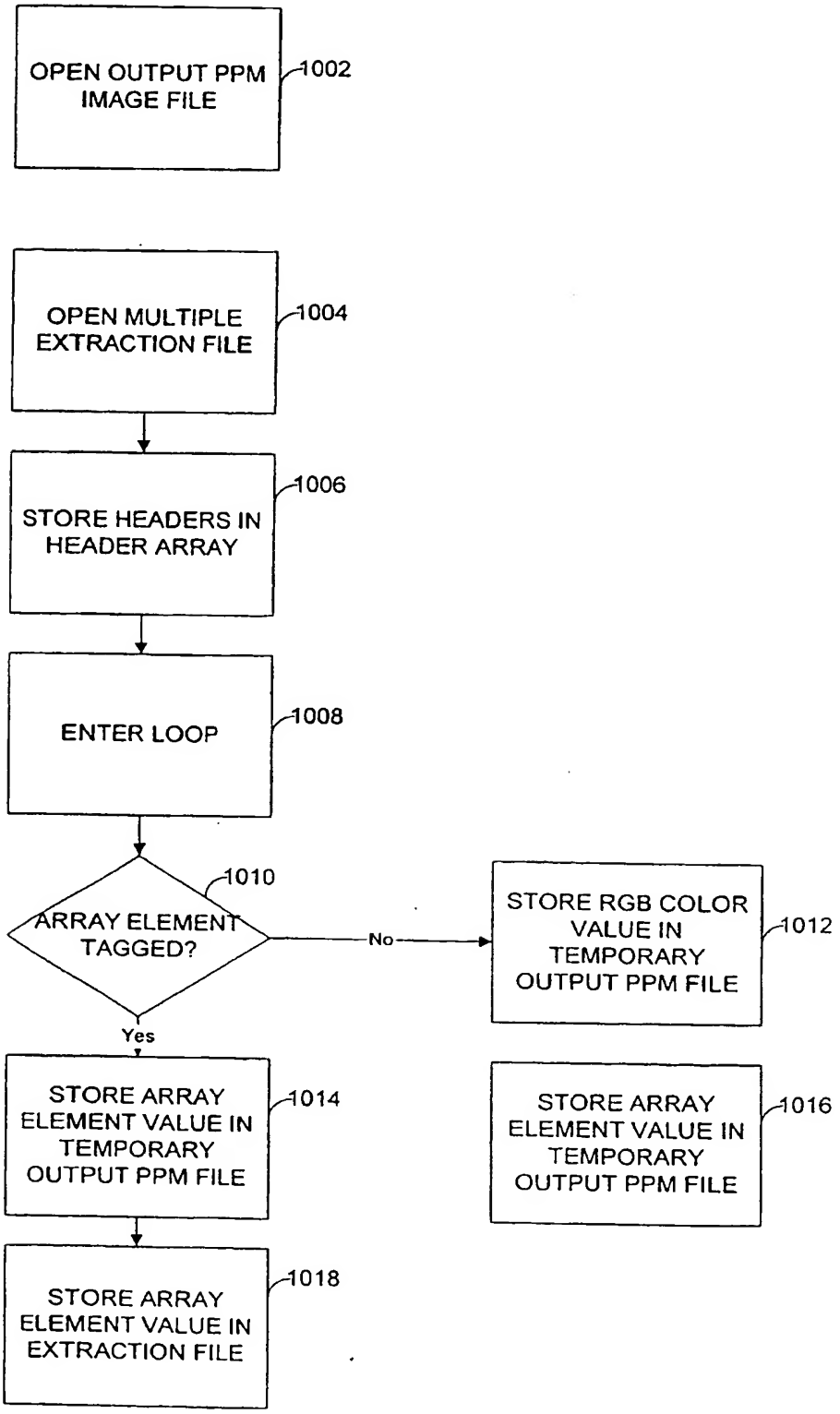
OPEN OUTPUT PPM IMAGE FILE
1002
OPEN MULTIPLE EXTRACTION FILE
1004
STORE HEADERS IN HEADER ARRAY
1006
ENTER LOOP
1008
ARRAY ELEMENT TAGGED?
1010
No
STORE RGB COLOR VALUE IN TEMPORARY OUTPUT PPM FILE
1012
Yes
STORE ARRAY ELEMENT VALUE IN TEMPORARY OUTPUT PPM FILE
1014
STORE ARRAY ELEMENT VALUE IN TEMPORARY OUTPUT PPM FILE
1016
STORE ARRAY ELEMENT VALUE IN EXTRACTION FILE
1018

FIGURE 10

FIGURE 11a

FIGURE 11b

INTERNATIONAL SEARCH REPORT

Interr. nal Application No
PCT/US 00/19900

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06T5/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06T

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, PAJ, INSPEC, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
P,A	US 5 995 115 A (DICKIE GARTH A) 30 November 1999 (1999-11-30) abstract column 2, line 10 - line 45 column 5, line 16 -column 6, line 7	1-22
A	MORTENSEN E N ET AL: "INTERACTIVE SEGMENTATION WITH INTELLIGENT SCISSORS" CVGIP GRAPHICAL MODELS AND IMAGE PROCESSING,US,ACADEMIC PRESS, DULUTH, MA, vol. 60, no. 5, 1 September 1998 (1998-09-01), pages 349-384, XP000782182 ISSN: 1077-3169 page 355, paragraph 3 page 356, paragraph 3.1	1-22

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
28 November 2000	05/12/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Gonzalez Ordonez, O

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Intern. nal Application No
PCT/US 00/19900

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 912 994 A (KLEIN LOUIS ET AL) 15 June 1999 (1999-06-15) abstract column 13, line 63 -column 14, line 2 column 15, line 30 -column 16, line 58	1-22

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

information on patent family members

Interr. nal Application No
PCT/US 00/19900

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5995115 A	30-11-1999	AU 6787298 A	30-10-1998
		EP 0972269 A	19-01-2000
		WO 9845809 A	15-10-1998
US 5912994 A	15-06-1999	NONE	